Ada COCOMO and the Ada Process Model

Barry Boehm and Walker Royce, TRW

1. Introduction

The orginal version of the Constructive Cost Model (COCOMO) [Boehm, 1981] was calibrated to 56 software development projects, and validated on 7 subsequent projects. On these projects, the COCOMO development effort estimates were accurate within 20% of the project actuals, about 70% of the time.

Subsequently, COCOMO has done approximately as well on most carefully collected sets of project data [Boehm, 1985; Jensen, 1985; Goudy, 1987]. In some cases, COCOMO has exhibited a systematic bias, but has done approximately as well once recalibrated to the specific environment [Boehm, 1985; Miyazaki-Mori, 1985; Marouane-Mili, 1989]. It should be noted that there are also some sets of project data for which COCOMO has been considerably less accurate [Kemerer,1987; Martin, 1988].

Recently, three software development approaches have motivated the development of a revised version of COCOMO: the use of the Ada programming language, the use of incremental development, and the use of an Ada process model capitalizing on the strengths of Ada to improve the efficiency of software development. This paper presents the portions of the revised Ada COCOMO dealing with the effects of Ada and the Ada process model.

The remainder of this section of the paper discusses the objectives of Ada COCOMO. Section 2 describes the Ada Process Model and its overall effects on software development effort and schedule. Section 3 presents the structure and features of Ada COCOMO, and discusses the rationale behind the changes made from the earlier version of COCOMO (called "standard COCOMO" in the remainder of the paper). Section 4 summarizes the current status of Ada COCOMO, including its calibration to date and its currently available implementations; and Section 5 presents the resulting conclusions.

Ada COCOMO Objectives

The primary objectives of the Ada COCOMO development activity were to:

- Determine the effect of Ada on software development costs and schedules. Some early studies of Ada's impact on software cost indicated that some of the cost driver factors might be different for Ada projects than for other projects, and that the phase distribution of cost and schedule might also be

different [Baskette, 1987]. We wished to determine whether this was the case and what, if any, changes needed to be made to COCOMO to accommodate Ada effects.

• Determine the effect of the Ada Process Model on software development costs and schedules. The Ada Process Model exploits some key Ada features (particularly, early compiler-checkable Ada package specifications and commonality of design-language and programming-language constructs), software risk management techniques [Boehm, 1989], and more general large-scale software engineering principles -- to provide a more efficient and controllable process model for software development. We wished to reflect the effects of this Ada process model in Ada COCOMO.

• Incorporate related COCOMO improvements. Since 1981, we have found several effects, primarily due to new technology, which have led to extensions to the original COCOMO cost drivers. We wished to incorporate these effects both into standard COCOMO and into Ada COCOMO.

2. The Ada Process Model

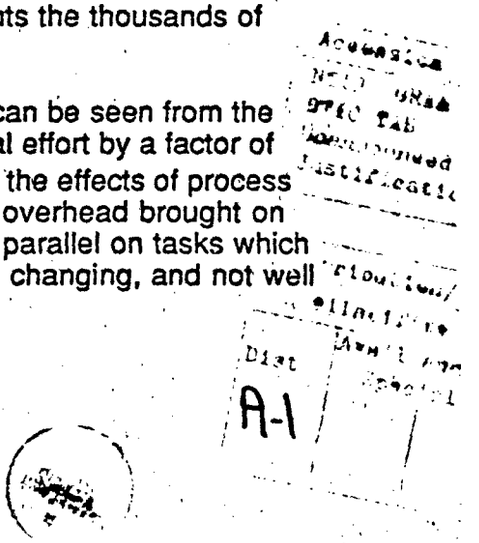
Earlier attempts to use software cost estimation insights to improve software productivity focused on improving the settings of software cost driver variables (via use of tools and modern programming practices, interactive workstations, removing hardware constraints, etc) and on reducing the amount of code one chooses to develop via reuse, fourth generation languages, requirements scrubbing, etc.) [Boehm, 1987].

The Ada Process Model attempts to further improve software productivity by reducing the exponent relating the size of the software product to the amount of effort required to develop it. For the COCOMO embedded mode (representing the type of challenging, real-time software projects primarily addressed by Ada), this equation is:

$$MM_{nom} = 2.8\ (KDSI)^{1.20},$$

where MM_{nom} represents the number of man-months required to develop an average or nominal software product, and KDSI represents the thousands of delivered source instructions in the product.

The resulting inefficiency or diseconomy of scale can be seen from the fact that doubling a product's size will increase its nominal effort by a factor of $(2)^{1.20} = 2.30$. The major sources of this inefficiency are the effects of process thrashing, turbulence, and interpersonal communication overhead brought on when large numbers of project personnel are working in parallel on tasks which are closely intertwined, incompletely defined, continually changing, and not well prepared for downstream integration.



Ada Process Model Strategy

The primary strategy elements the Ada Process Model uses to reduce these inefficiencies or diseconomies of scale are to:

- Produce compilable, compiler-checked Ada package specifications (and body outlines), expressed in a well-defined Ada Design Language (ADL), for all top-level and critical lower-level Ada packages, by the project's or increment's Preliminary Design Review (PDR).

- Identify and eliminate all major risk items by PDR. This has the effect of focusing the PDR (and other early reviews as well) on working demonstrations of prototypes or kernel capabilities rather than on large amounts of paper.

- Use a phased incremental development approach, with the requirements for each increment, called a build, stabilized by the build's PDR.

These three conditions minimize project diseconomies of scale by eliminating the following primary sources of software project inefficiency and turbulence:

1. Interpersonal Communications Overhead. An issue requiring the coordination of N agents requires the exercise of N(N-1)/2 communication paths. The Ada Process Model minimizes this effect by keeping the design team size N small until PDR, and by establishing Ada package specifications for unit-level interfaces by PDR, thus minimizing the number of issues the larger post-PDR development team will have to coordinate.

2. Late Rework. Even with rigorous package specifications, a project will lapse into turbulence if the resolution of a high-risk "architecture breaker" problem requires redefinition of many of the package specifications. Eliminating such risk items by PDR ensures that the project may proceed efficiently with its initially-determined package specifications.

3. Unstable requirements. Even with rigorous package specifications and no post-PDR risk items, a project will lapse into turbulence if there is a continuing stream of requirements changes impacting the definition of the package specifications. Simply raising the threshold of allowable requirements changes can reduce this turbulence considerably. Incremental development reduces the turbulence effects even further by reducing the amount of software under development at any given time, and by enabling the deferral of requirements changes to downstream increments.

Ada Process Model Overview and Features

An overview of the Ada Process Model is shown as Figure 1. Additional features of the Ada Process Model include the use of:

4. Small up-front system engineering and design teams, with expertise in software architecture, Ada, and the applications domain. Such people are a software project's scarcest resource. The Ada Process Model optimizes their contribution, and by providing validated package specifications to the larger number of later developers, allows these positions to be staffed by more junior people.

5. A project risk management plan to determine the approach for eliminating risk items by PDR, and also to determine the sequence of development increments. Early increments focus on development of "executing architecture skeletons" to ensure that critical system nuclei are satisfactorily implemented early. They also focus on prototypes to eliminate risks associated with user interface uncertainties, critical algorithms, or incorporation of state-of-the-art computer science capabilities. Middle increments flesh out the higher-priority, better-understood product capabilities. Later increments provide additional functions as their needs become better understood.

6. An expanded incremental development approach involving lower increment levels within each build: build increments, which reflect the planned order of development of each build; and component increments, which represent increments of transition between Ada Design Language (ADL) and Ada code for individual components.

7. Intermediate technical walkthroughs in the early requirements and design phases. These focus the pre-walkthrough effort on problem-solving and architecture definition, and the post-walkthrough effort on document production.

8. Individual detailed design walkthroughs for each component instead of a massive Critical Design Review (CDR)> Instead, an efficient CDR is held to cover the highlight issues of the walkthroughs.

9. Continuous integration via compiler checking of Ada package specifications and continuous expansion of ADL statements into Ada code, rather than beginning integration at the end of unit test.

10. Bottom-up requirements verification via unit standalone tests, build integration tests, and engineering string tests, so that the demonstration of requirements satisfaction has been mostly done by the end of system test.

11. Well-commented Ada code and big-picture design information instead of massive as-built Software Detailed Design Documents, which rapidly get out of date and lose their maintenance value.

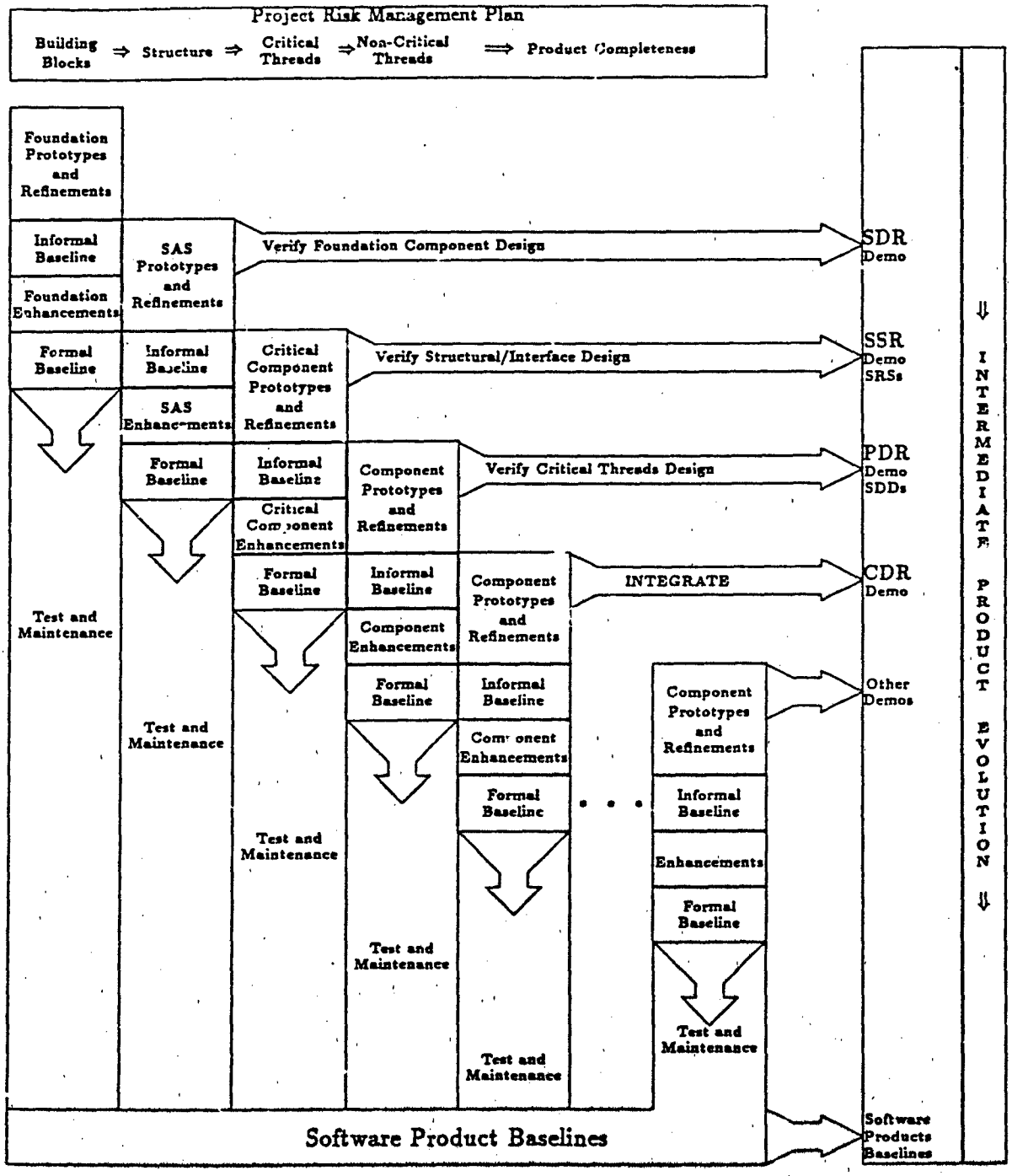


Figure 1: Incremental Development Under The Ada Process Model

12. A set of automated metrics tightly coupled to the project's Software Development Plan and its build definitions. Conventions on ADL and its expansion into Ada code enable metric tools to provide detailed visibility into the usually-obscure code development process.

The resulting Ada Process Model can be and has been used successfully as a tailored version of such Government standards as DoD-STD-2167. This was done initially on a small TRW-internal project, and is currently underway on a large Air Force project: the Command Center Processing and Display System-Replacement (CCPDS-R) project. CCPDS-R is currently about 24 months into its 36-month development schedule. It has completed 3 of its 5 builds (about 300,000 of its planned 500,000 lines of Ada), and has met all of its Ada Process Model milestones to date. More details on the Ada Process Model and its application to CCPDS-R are given in [Royce, 1989]. Although the model was developed for use on Ada projects and has some Ada-specific features, many of its features can be applied to non-Ada projects as well.

Cost and Schedule Implications of the Ada Process Model

Because of the reduction in project communications overhead and diseconomies of scale, the use of the Ada Process Model leads to an overall reduction in project effort. The overall schedule for a single-shot development is lengthened somewhat, but the use of incremental development means that users receive their initial operating capability earlier. The phase distribution of effort and schedule also changes. Use of the Ada Process Model involves more effort and schedule for requirements analysis and design, and considerably less for code, integration, and test.

Ada COCOMO models the effect of the Ada Process Model in terms of a reduction in the nominal effort equation exponent of 1.20. The new nominal effort equation takes the form:

$$MM_{nom} = 2.8\ (KDSI)^{1.04+\Sigma}$$

The parameter Σ measures the project's estimated degree of compliance with the Ada Process Model in terms of four parameters:

- The percent of the design that has been expressed as compiler-checked Ada package specifications and body outlines by PDR.
- The percent of the risk items that have been eliminated by PDR.
- The degree to which the requirements have been stabilized by PDR.
- The team's previous experience in applying the Ada Process Model.

If a project is fully compliant with the Ada Process Model, then Σ will be 0.00, and the diseconomy of scale exponent will be 1.04. If a project exhibits the current typical hasty-PDR symptoms, then Σ will be 0.16, and the exponent will be 1.20, the same as for the current COCOMO embedded-mode model. The Σ parameter is also used in Ada COCOMO in estimating the development schedule and the phase distribution of effort and schedule.

3. Ada COCOMO Structure and Features

This Section provides the information necessary to use the essential portions of Ada COCOMO or to implement them in a computer program. Section 3.1 summarizes the differences between Ada COCOMO and standard COCOMO. Section 3.2 provides a structural overview of the model's computations. Section 3.3 provides the information necessary to determine the project's Ada Process Model Σ parameter and to determine the project's nominal effort estimate. Section 3.4 provides the rating scales and effort multipliers for the Ada COCOMO cost driver variables. Section 3.5 provides the Ada COCOMO schedule estimation equation, the tables for determining the phase distribution of project effort and schedule, and an overview of the incremental development model. Section 3.6 provides an example comparing Ada COCOMO and standard COCOMO estimates on two sample projects.

3.1 Differences Between Ada COCOMO and Standard COCOMO

Ada COCOMO has three categories of differences from standard COCOMO:

a. General improvements to COCOMO, which can be incorporated as improvements to standard COCOMO as well. These comprise a wider range of ratings and effects due to software tools and turnaround time; the splitting of virtual machine volatility effects into host and target machine effects; the elimination of added costs due to schedule stretchout; the addition of cost drivers to cover effects of security-classified projects and development for software reusability; and the addition of a model for incremental development.

b. Ada-specific effects, including reduced multiplier penalties for higher levels of required reliability and product complexity; a wider range of ratings and multipliers for programming language experience; and a set of Ada-oriented instruction-counting rules, including the effects of software reuse in Ada.

c. Effects of using the Ada Process Model, which can largely be adapted to projects using other programming languages. Their use on non-Ada projects would require some experimental tailoring of standard COCOMO to accommodate the resulting cost and schedule effects. These effects include the revised exponential scaling equations for

nominal development effort, development schedule, and nominal maintenance effort; the extended range of modern programming practices effects; the revised ranges of analyst capability and programmer capability effects; and the revised phase distributions of effort and schedule.

The remainder of standard COCOMO remains the same as it was: the overall functional form, most of the effort multipliers, the software adaptation equations, the activity distribution tables, and the use of annual change traffic for software maintenance estimation. Standard COCOMO also covers all three COCOMO development modes; to date, there is only an Ada COCOMO counterpart of the COCOMO Embedded mode.

3.2 Ada COCOMO Structural Overview

Figure 2 provides an overview of the Ada COCOMO steps used to estimate software development costs and schedules. The first step uses estimates of the software size in thousands of delivered source instructions (KDSI) and the Ada Process Model Σ factor to calculate the nominal man-month estimate: the amount of effort the project would require if it were perfectly average in all respects. Step 2 involves determining the cost-sensitive ways in which the project is different from average. The project is rated in terms of 18 cost driver attributes; the ratings are used to determine a set of 18 effort multipliers; these are multiplied together and applied to the nominal man-month estimate to produce the project's estimated effort in man-months. At this point, a cost per man-month figure may be applied to determine the project's cost in dollars or other currencies.

Step 3 estimates the project's development schedule (from its software requirements review to its software acceptance test) as a function of its estimated man-months and its Σ factor. Step 4 estimates the phase distribution of effort and schedule from a set of tables of distribution percentages vs size. In Step 5, the effects of incremental development can be estimated by repeatedly applying Steps 1-4 to the overall project and to the individual increments, and appropriately phasing the increments' estimated budgets and schedules.

3.3 The Ada Process Model Σ Factor

Figure 3 shows the rating scale used to determine a project's Ada Process Model Σ factor. Each of the four elements has a rating scale from 0.00 to 0.05; the ratings for each element are added together to define Σ. The first element, Experience with the Ada Process Model, is easier to rate in advance of the project than the others: one determines the degree of experience with the model of the project's key personnel, and uses this to determine this element's numerical contribution to Σ.

Figure 2.

Ada COCOMO STRUCTURAL OVERVIEW

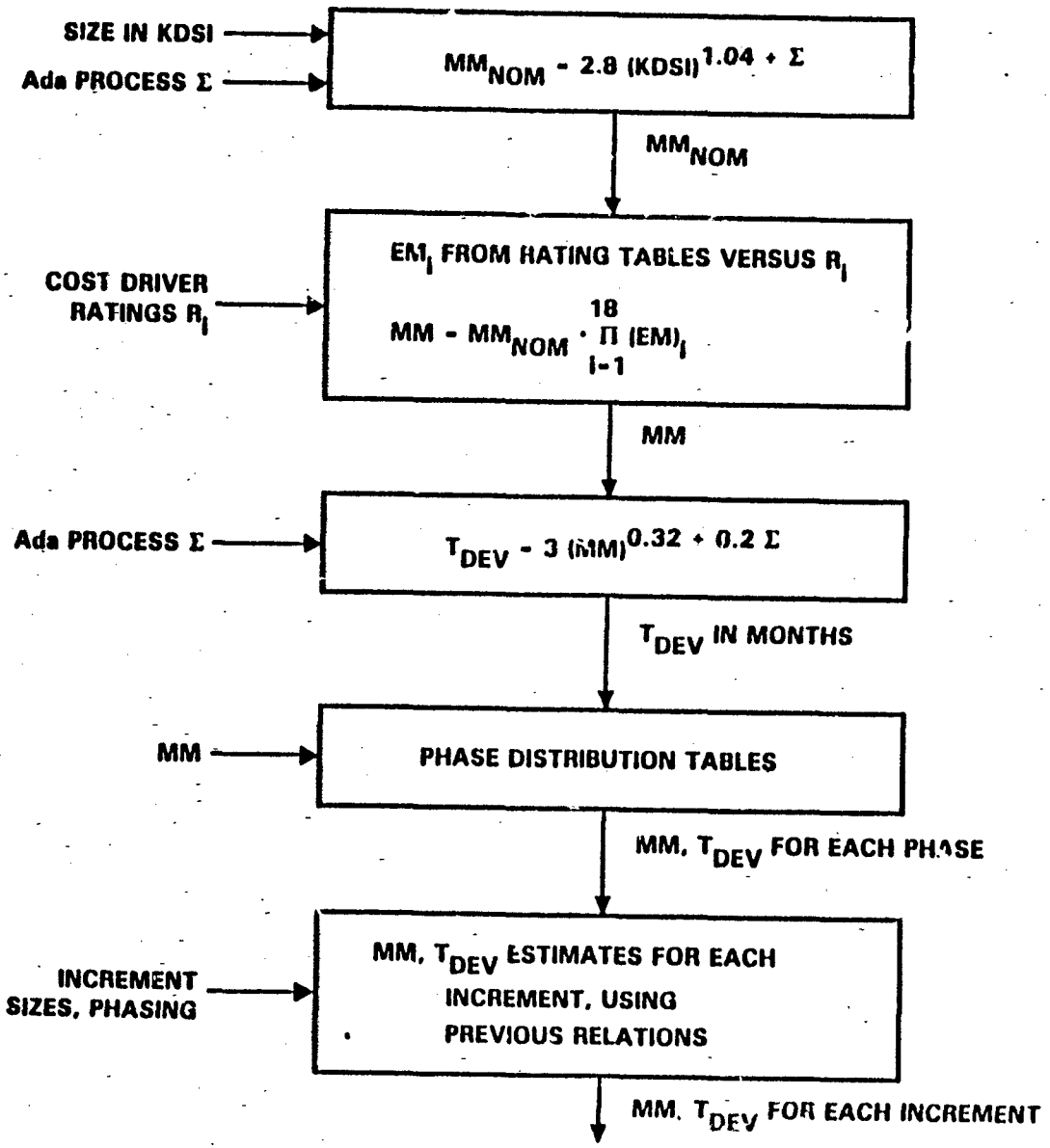


Figure 3.

Ada COCOMO SCALING EQUATION

- $MM_{NOM} = (2.8)\,(KDSI)^{1.04 + \sum_{i=1}^{4} W_i}$

WEIGHTS W_i	0.00	0.01	0.02	0.03	0.04	0.05
EXPERIENCE WITH Ada PROCESS MODEL	SUCCESSFUL ON >1 MISSION-CRITICAL PROJECT	SUCCESSFUL ON 1 MISSION-CRITICAL PROJECT	GENERAL FAMILIARITY WITH PRACTICES	SOME FAMILIARITY WITH PRACTICES	LITTLE FAMILIARITY WITH PRACTICES	NO FAMILIARITY WITH PRACTICE
DESIGN THOROUGHNESS AT PDR (UNIT PACKAGE SPECS COMPILED, BODIES OUTLINED)	FULLY (100%)	MOSTLY (90%)	GENERALLY (75%)	OFTEN (60%)	SOME (40%)	LITTLE (20%)
RISKS ELIMINATED BY PDR	FULLY (100%)	MOSTLY (90%)	GENERALLY (75%)	OFTEN (60%)	SOME (40%)	LITTLE (20%)
REQUIREMENTS VOLATILITY DURING DEVELOPMENT	NO CHANGES	SMALL NONCRITICAL CHANGES	FREQUENT NONCRITICAL CHANGES	OCCASIONAL MODERATE CHANGES	FREQUENT MODERATE CHANGES	MANY LARGE CHANGES

ALL 0.04'S: COCOMO EMBEDDED MODE

- TYPICAL MM_{NOM}'S

ΣW_i	0.00	0.04	0.08	0.12	0.16	0.20
100 KDSI	336	405	487	585	703	846
500 KDSI	1,795	2,302	2,951	3,784	4,852	6,221

For the other elements, Figures 4-6 provide a more detailed set of rating scales which can be applied early in the project's life cycle. Let us look at an example project in terms of Figure 4, which provides several criteria for estimating a project's likely level of design thoroughness by PDR.

Suppose that the example project has a relatively sketchy risk management plan, so that the best that could be claimed in Row 1 of Figure 4 is "some" compatibility with it (an 0.03 rating). On the other hand, the project has 25% of its development schedule devoted to the preliminary design phase (an 0.02 rating); 100% of the necessary architects available (an 0.01 rating); and good front-end tool support: a solid Ada compiler, but no related graphic architecture and design aids (an 0.02 rating). The level of uncertainty in key architecture drivers could be rated as either "some" (0.02) or "considerable" (0.03), but since the other items average out to 0.02, using an 0.02 rating for the Design Thoroughness by PDR element would be reasonable.

The project would use Figures 5 and 6 in a similar fashion to determine Σ ratings for these elements. Suppose that the Risk Elimination by PDR rating from Figure 5 was 0.03, the Requirements Volatility rating from Figure 6 was 0.01, and the rating for Ada Process Model familiarity was 0.02. Adding these ratings to the Design Thoroughness rating of 0.02 gives an overall Σ factor of 0.08. Therefore, the nominal effort equation exponent for this project would be 1.04 + 0.08 = 1.12; the resulting equation is:

$$MM_{nom} = 2.8 \text{ (KDSI)}^{1.12}.$$

The data at the bottom of Figure 3 show how significant an influence the Σ factor has upon software costs. For a nominal 100,000 instruction product, a Σ = 0.08 project is estimated to require 487 man-months for development. A Σ = 0.16 rating would increase this estimate to 703 MM, while a decrease to Σ = 0.00 would decrease the estimated effort to 336 MM.

While the overall effect of the Σ factor is significant, it does not introduce instabilities in Ada COCOMO. A change of one rating level on one of the rating scales in Figure 3 corresponds to a change of about 5% in the estimated development effort for a 100 KDSI product.

3.4 Ada COCOMO Cost Driver Variables

The differences between the Ada COCOMO cost driver variables and their COCOMO counterparts are:

- Two new variables have been added: Required Reusability (RUSE) and Classified Security Application (SECU).
- The Virtual Machine Volatility (VIRT) variable has been split into host volatility (VMVH) and target volatility (VMVT) effects.

Figure 4.
Ada COCOMO Σ Factor: Design Thoroughness by PDR

Characteristic \ Σ Rating	.00	.01	.02	.03	.04	.05
• Schedule, budget, and internal milestones through PDR compatible with Risk Management Plan	Fully	Mostly	Generally	Some	Little	None
• Percent of development schedule devoted to Preliminary Design phase	40	33	25	17	10	5
• Percent of required top software architects available to project	120	100	80	60	40	20
• Tool support for developing and verifying ~~Ada packages specs~~ SAS components	Full	Strong	Good	Some	Little	None
• Level of uncertainty in key architecture drivers: mission, user interface, hardware, COTS, technology, PERFORMANCE	Very Little	Little	Some	Consid-erable	Signif-icant	Extreme

Figure 5.

Ada COCOMO Σ Factor: Risk Elimination by PDR

Characteristic \ Σ Rating	.00	.01	.02	.03	.04	.05
• Risk Management Plan identifies all critical risk items, establishes milestones for resolving them by PDR	Fully	Mostly	Generally	Some	Little	None
• Schedule, budget, and internal milestones through PDR compatible with Risk Management Plan	Fully	Mostly	Generally	Some	Little	None
• Percent of development schedule devoted to Preliminary Design phase	40	33	25	17	10	5
• Percent of required top software architects available to project	120	100	80	60	40	20
• Tool support available for resolving risk items	Full	Strong	Good	Some	Little	None
• # of functional RISKS # of Risks w/ potential for major impact (SW)	~~≤1~~	~~2-3~~	4 - 6	6 - 10	10 - 15	> 15
~~# of Rs~~						

Something like this seems necessary

A project should be able to estimate the number of key SW risks a priori

Figure 6.
Ada COCOMO Σ Factor: Requirements Volatility

Characteristic \ Σ Rating	.00	.01	.02	.03	.04	.05
• System requirements baselined, under rigorous change control	Fully	Mostly	Generally	Some	Little	None
• Level of uncertainty in key requirements areas: mission, user interface, hardware, other interfaces	Very Little	Little	Some	Considerable	Significant	Extreme
• Organizational track record in keeping requirements stable	Excellent	Strong	Good	Moderate	Weak	Very Weak
• Use of incremental development to stabilize requirements	Full	Strong	Good	Some	Little	None
• System architecture modularized around major sources of change	Fully	Mostly	Generally	Some	Little	None

- Several variables have new effort multipliers and/or new rating levels.
- Complementary changes have been incorporated in the Ada COCOMO maintenance estimation model.

Figure 7 shows the overall comparison between the COCOMO and Ada COCOMO cost driver multiplier ranges for the Intermediate level of COCOMO. The corresponding numerical multipliers for Intermediate Ada COCOMO are given in Table 1. The corresponding multipliers and rating scales for Detailed Ada COCOMO and the maintenance portion of Ada COCOMO are available in [Boehm-Royce, 1987]; within the confines of this paper, we will focus on the differences in Intermediate Ada COCOMO and their rationale.

New Cost Driver Variables

The Required Reusability variable (RUSE) has been added to address the effects of developing software for reuse in future situations. The rating scales and their corresponding effort multipliers (E.M) are:

RUSE Rating	E.M	Rating Description
Nominal	1.0	No reuse
High	1.10	Reuse within single-mission products
Very High	1.30	Reuse across a single product line
Extra High	1.50	Reuse in any application

The added effort for more general reusability levels reflects the need for more generic design of the software, more elaborate documentation, and more extensive testing to ensure confident reuse in all specified situations. The cost of incorporating the reusable software into a product is handled in the same way as in standard COCOMO, via estimation of the amount of adapted software and the percentages of change required in its design, code, and integration to incorporate it into the new product.

The Classified Security Application variable (SECU) has been adapted from the discussion in Chapter 28 of [Boehm, 1981]. The rating scales and effort multipliers are:

SECU Rating	E.M	Rating Description
Nominal	1.0	Unclassified Project
High	1.10	Classified (Secret or Top Secret) Project

Host-Target Effects

The increased use of the host-target mode of software development has led to a split in the Virtual Machine Volatility (VIRT) cost driver variable into two

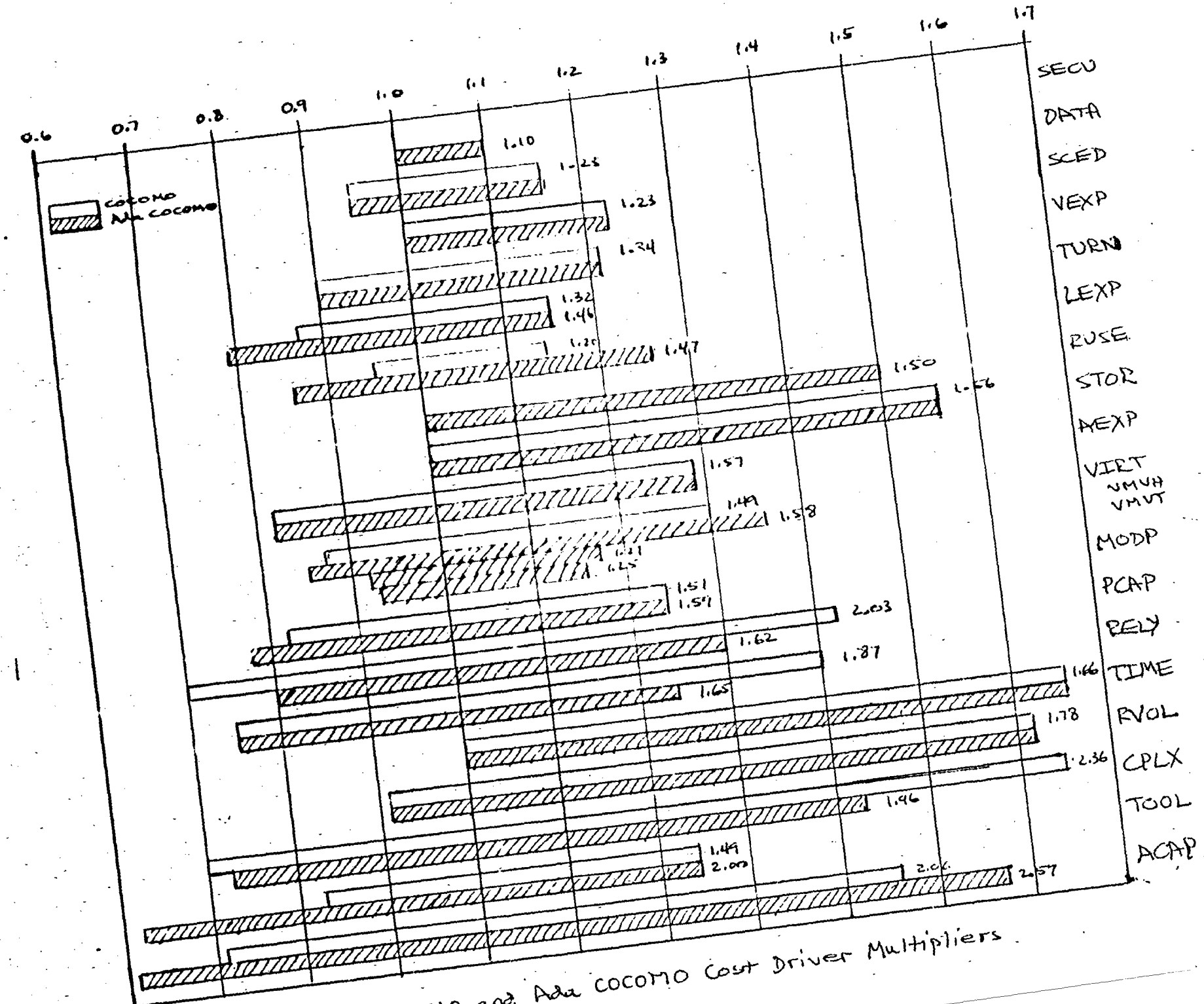


Figure 7. COCOMO and Ada COCOMO Cost Driver Multipliers

Table 1.

SUMMARY OF IOC Ada COCOMO

EFFORT: $MM = 2.8\ (KDSI)^{1.04 + \sum_{i=1}^{4} w_i} \left(\prod_{i=1}^{18} (EM_i)\right)$

SCHEDULE: $T_{DEV} = 3.0\ (MM)^{0.32 + 0.2 \sum_{i=1}^{4} w_i}$

EFFORT MULTIPLIERS	VERY LOW	LOW	NOM.	HIGH	VERY HIGH	EXTRA HIGH	XX HIGH	RANGE
RELY	0.75	0.88	0.96	1.07	1.24			1.65
DATA		0.94	1.0	1.08	1.16			1.23
CPLX	0.73	0.85	0.97	1.08	1.22	1.43		1.96
RUSE			1.0	1.10	1.30	1.50		1.50
TIME			1.0	1.11	1.30	1.66		1.66
STOR			1.0	1.06	1.21	1.56		1.56
VMVH		0.92	1.0	1.09	1.17			1.27
VMVT		0.93	1.0	1.07	1.16			1.25
TURN	0.79	0.87	1.0	1.07	1.15			1.46
ACAP	1.57	1.29	1.0	0.80	0.61			2.57
PCAP	1.30	1.12	1.0	0.89	0.80			1.62
AEXP	1.29	1.13	1.0	0.91	0.82			1.57
VEXP	1.21	1.10	1.0	0.90				1.34
LEXP	1.26	1.14	1.04	0.95	0.86			1.47
MODP	1.24	1.10	0.98	0.86	0.78			1.59
TOOL	1.24	1.10	1.0	0.91	0.83	0.72	0.62	2.00
SCED	1.23	1.08	1.0	1.0	1.0			1.23
SECU			1.0	1.10				1.10

variables: Virtual Machine Volatility - Host (VMVH) and Virtual Machine Volatility - Target (VMVT). The rating scales and corresponding effort multipliers are:

Rating	VMVH	VMVT	VMVH*VMVT	VIRT
Low	0.92	0.93	0.86	0.87
Nominal	1.00	1.00	1.00	1.00
High	1.09	1.07	1.17	1.15
Very High	1.17	1.16	1.36	1.30
Prod'y Range	1.27	1.25	1.58	1.49

The Productivity Range is defined as the ratio between the highest and the lowest effort multipliers for any given cost driver variable. It is a measure of the leverage that the variable has on software development costs. The combined productivity range for VMVH and VMVT is slightly higher than the corresponding productivity range for the COCOMO VIRT variable. This reflects the deeper interaction between operating system services and Ada programs (e.g., via tasking and exceptions) than with earlier programming languages.

New Effort Multipliers: Ada Effects

The Ada COCOMO variables Required Reliability (RELY), Product Complexity (CPLX), Language Experience (LEXP), and Use of Modern Programming Practices (MODP) have different multipliers in Ada COCOMO than in standard COCOMO due to effects of the Ada programming language.

For the Nominal, High, and Very High rating levels of Required Reliability, the effort multipliers are reduced in Ada COCOMO. This is due primarily to such Ada language features as strong typing, tasking, exceptions, and packages, which prevent many classes of software errors from occurring, limit the side effects of errors, and make some classes of errors easier to find. The comparison with standard COCOMO is shown below:

RELY Rating	Ada COCOMO	COCOMO E.M
Very Low	0.75	0.75
Low	0.88	0.88
Nominal	0.96	1.00
High	1.07	1.15
Very High	1.24	1.40
Prod'y Range	1.65	1.87

For Product Complexity, several of the rating levels have different effort multipliers:

CPLX Rating	Ada COCOMO	COCOMO E.M
Very Low	0.73	0.70
Low	0.85	0.85
Nominal	0.97	1.00
High	1.08	1.15
Very High	1.22	1.30
Extra High	1.43	1.65
Prod'y Range	1.96	2.36

The Ada COCOMO effort multiplier for Very Low complexity is slightly higher than its COCOMO counterpart due to the extra work required in Ada for very simple programs. The effort multipliers for the higher rating levels are lower than their COCOMO counterparts since there are a number of Ada paradigms (tasking, exceptions, record types, access types) which make previously complex programming constructs more straightforward to implement.

The Programming Language Experience variable (LEXP) in Ada COCOMO has an additional rating level and a significantly wider productivity range than its standard COCOMO counterpart. The comparison is shown below:

LEXP Rating	Ada COCOMO	COCOMO E.M
Very Low	1.26	1.14
Low	1.14	1.07
Nominal	1.00	1.04
High	0.95	0.95
Very High	0.86	
Prod'y Range	1.47	1.20

The Very High rating corresponds to an Ada experience level of at least 6 years. The main reasons for the difference from standard COCOMO is that Ada is a much richer and more complex language than most previous languages. (An exception is PL/I, which has had a similar widened productivity range for LEXP in the Jensen model [Jensen-Lucas, 1985]). For many computer program functions, there are several ways to implement them in Ada, each with somewhat different side effects (e.g., branches, case statements, loops, tasking statements, exceptions, procedure calls). An experienced Ada developer can capitalize on this richness to simplify many program functions, while an inexperienced Ada developer is more likely to choose an Ada implementation with harmful side effects (performance problems, reliability problems, modifiability problems, difficulties in handling nonstandard conditions, or just plain errors).

The Ada COCOMO Use of Modern Programming Practices variable (MODP) has a slightly different set of effort multipliers than standard COCOMO,

due to Ada's improved support of modern programming practices, particularly modularity and information hiding [Parnas, 1979] and object-oriented development [Booch, 1987]. The resulting comparison is as follows:

MODP Rating	Ada COCOMO	COCOMO E.M
Very Low	1.24	1.24
Low	1.10	1.10
Nominal	0.98	1.00
High	0.86	0.91
Very High	0.78	0.82
Prod'y Range	1.59	1.51

For maintenance, MODP is treated differently in Ada COCOMO, in that MODP ratings are used to determine a large component of the Σ size-scaling factor, rather than to determine effort multipliers as a function of size. Details are provided in [Boehm-Royce, 1987].

New Effort Multipliers: Ada Process Model Effects

The Ada Process Model's emphasis on achieving a solid and stable architecture for the software product's life-cycle reduces the damage that less-capable programmers can do to the project. But it also places a heavy reliance on the capability of the analysts to create such an architecture. Thus, Ada COCOMO has a different set of effort multipliers for Analyst Capability (ACAP) and Programmer Capability (PCAP) than standard COCOMO. The comparison between full use of the Ada Process Model (Σ=0.00) and standard COCOMO or non-use of the Ada Process Model is as follows:

Rating	Full Ada P.M. (Σ=0.00)			Std. COCOMO (Σ=0.16)		
	ACAP	PCAP		ACAP	PCAP	
Very Low	1.57	1.30		1.46	1.42	
Low	1.29	1.12		1.19	1.17	
Nominal	1.00	1.00		1.00	1.00	
High	0.80	0.89		0.86	0.86	
Very High	0.61	0.80		0.71	0.70	
Prod'y Range	2.57	1.62	4.16	2.06	2.03	4.14

Although the full use of the Ada Process Model correlates with a wider productivity range for Analyst Capability and a narrower productivity range for Programmer Capability, their combined productivity range is essentially the same as for standard COCOMO (4.16 vs.4.14).

For partial use of the Ada Process Model (Σ between 0.00 and 0.16), one interpolates between the values of ACAP and PCAP for full use and non-use of

the Ada Process Model. For example, a project at the half-way point (Σ=0.08) with Very High capability analysts and programmers would use the following effort multipliers:

ACAP: 0.61 + (0.08 / 0.16) (0.71 - 0.61) = 0.66.

PCAP: 0.80 + (0.08 / 0.16) (0.70 - 0.80) = 0.75.

New Effort Multipliers: General Effects

The Ada COCOMO variables Computer Turnaround Time (TURN), Use of Software Tools (TOOL), and Required Development Schedule (SCED) differ from their standard COCOMO counterparts due to general technology effects which are independent of Ada. The TURN variable has an additional rating level of Very Low, reflecting the more effective exploitation of interactive software development as compared to the late 1970's, when standard COCOMO was calibrated. The main changes are:

- Furnishing every software project person with an interactive terminal, as compared to the late-1970's average of roughly 0.3 terminals per project person.
- Interactive support of all project phases, as compared to primary support of the code and unit test phase in the late 1970's.

The resulting comparison of TURN effort multipliers is as follows:

TURN Rating	Ada COCOMO	COCOMO E.M
Very Low	0.79	
Low	0.87	0.87
Nominal	1.00	1.00
High	1.07	1.07
Very High	1.15	1.15
Prod'y Range	1.46	1.32

The TOOL variable has two additional rating levels, Extra High and XX-High, reflecting more fully populated and integrated tool sets than were available in the late 1970's. The Extra High level reflects a partly integrated tool set, using a Unix-level (pipes and hierarchical ASCII files) ~~level~~ of tool interoperability. The XX-High level has not yet been fully achieved by a software engineering environment. One way of visualizing its level of capability is that it would provide for all software project life cycle functions the level of incremental analysis and feedback capability that the current Rational environment provides for incremental Ada syntax and semantic analysis.

The resulting comparison of TOOL effort multipliers is as follows:

TOOL Rating	Ada COCOMO	COCOMO E.M.
Very Low	1.24	1.24
Low	1.10	1.10
Nominal	1.00	1.00
High	0.91	0.91
Very High	0.83	0.83
Extra High	0.73	
XX High	0.62	
Prod'y Range	2.00	1.49

The SCED variable has been changed to eliminate the effort penalty associated with stretching a software project's schedule beyond the "natural" schedule estimated by the Ada COCOMO schedule equation. This branch of the effort-vs.-schedule tradeoff curve has been a point of difference between software cost models. The SLIM [Putnam, 1978], Jensen [Jensen-Lucas, 1983], and Softcost [Reifer, 1988] models incorporate a large, unending effort decrease as schedule is stretched; the standard COCOMO and Price S [Freiman-Park, 1979] models have incorporated a slight effort increase as schedule is stretched. The rationale in Ada COCOMO for keeping the High and Very High schedule-stretch multipliers at 1.00 is that incremental development has not been observed to incur a cost penalty, if the incremental strategy has been well planned out in advance, although it will stretch the development schedule.

Unchanged Cost Driver Variables

The remaining COCOMO cost driver variables were unchanged in Ada COCOMO, reflecting the judgement that their effort multiplier effects were highly independent of the use of Ada. These were Data Base Size (DATA), Execution Time Constraint (TIME), Main Storage Constraint (STOR), Applications Experience (AEXP), and Virtual Machine Experience (VEXP). An overall summary of the Ada COCOMO effort multipliers is given in Table 1.

Detailed COCOMO and Maintenance Effort Multipliers

In addition, Ada COCOMO has compatible phase-specific effort multipliers for implementing the Detailed version of COCOMO, and counterpart changes in the maintenance effort multipliers for the Required Reliability (RELY) variable. Details are provided in [Boehm-Royce, 1987].

3.5 Other Ada COCOMO Changes

Schedule Estimation

The standard COCOMO embedded-mode schedule equation is:

$$T_{DEV} = 2.5\,(MM)^{0.32},$$

where T_{DEV} is the development time in months from the Software Requirements Review to the Software Acceptance Test, and MM is the estimated development effort in man-months. The corresponding Ada COCOMO schedule equation is:

$$T_{DEV} = 3.0\,(MM)^{0.32+0.2\Sigma},$$

where Σ is the Ada Process Model compliance factor.

The revised coefficient and added Σ factor were determined from calibration to the two Ada projects originally used to calibrate Ada COCOMO. The equation has also been a good fit to three subsequent Ada projects. The equation indicates a longer development time for a given man-month level due to longer early schedule investments in requirements and design definition and validation. These are compensated by the overall reduction in man-months from using the Ada Process Model, and by the use of incremental development, which provides an initial usable increment earlier than a single-shot development of the entire product can provide.

Phase Distribution of Effort and Schedule

The Ada Process Model requires more effort and schedule in the early phases of software development, and saves considerably more effort and schedule in the later phases. This is borne out by the comparison of phase distributions in Table 2 between full use of the Ada Process Model ($\Sigma = 0.00$) and non-use of the Ada Process Model (the standard COCOMO Embedded Mode, or $\Sigma = 0.16$). A project which partially complies with the Ada Process Model would therefore interpolate in Table 2 to find its phase distribution of effort and schedule. Thus, for example, a 128-KDSI Embedded Mode project with a Σ of 0.08 would have phase distributions halfway between the table entries for a 128-KDSI project: e.g., 20.5% of its effort and 37.5% of its schedule in the Product Design phase.

The use of the Ada Process Model has also changed the definition of two of the major milestones defining the phase endpoints. Table 3 shows the revised definitions of the Software Requirements Review and the Preliminary (or Product) Design Review; the other milestones are largely the same as for standard COCOMO.

Incremental Development

Table 2.
Ada COCOMO : Phase Distribution of Effort and Schedule

X-Y: Full Use - No Use of Ada Process Model : Embedded Mode Projects
$\Sigma = 0.0 - \Sigma = 0.16$

Effort Distribution / Phase	SIZE				
	Small 2 KDSI	Intermediate 8 KDSI	Medium 32 KDSI	Large 128 KDSI	Very Large 512 KDSI
Plans and Requirements (%)	12-8	12-8	12-8	12-8	12-8
Product Design	23-18	23-18	23-18	23-18	23-18
Programming	57-60	55-57	53-54	51-51	49-48
Detailed Design	32-28	31-27	30-26	29-25	28-24
Code and Unit Test	25-32	24-30	23-28	22-26	21-24
Integration and Test	20-22	22-25	24-28	26-31	28-34
Schedule Distribution					
Plans and Requirements	28-24	32-28	36-32	40-36	44-40
Product Design	33-30	35-32	37-34	39-36	41-38
Programming	52-48	48-44	44-40	40-36	36-32
Integration and Test	15-22	17-24	19-26	21-28	23-30

Table 3a.

Software Requirements Review Milestone

System Architecture and CSCI Definitions

- External and inter-CSCI interfaces
- Top-level CSCI and inter-CSCI control and data flows, behavioral representation, object definitions
- System level data dictionary

Individual CSCI Requirements

- Functional, performance, and quality requirements
- Directions of growth and change identified
- Traceability to system requirements
- Level of detail proportional to risk exposure

ADD SSPM

Results of system level risk identification and risk reduction activities

Completed Software Development Plan, Including Risk Management Plan

Software Life Cycle Plan

- Key provisions for training, conversion, installation, operations, and support

Product Control Plans

- Detailed configuration management and quality assurance plans
- Overall verification and validation plan (excluding detailed test plans)

Table 5b.

Preliminary Design Review Milestone

PDW	PDR1	
*	*	Overall software architecture defined to CSC level
*	*	Overall mapping of CSC's to builds with rationale
*	*	All major risk items identified and resolved
*	*	~~Executing architecture kernel~~ JAS Baseline & Demonstration of Key Critical Threads
*		Verified preliminary design specification for this application build
		- Top level CSC and critical lower level CSC specs completed and verified; bodies outlined
		- Physical and logical data structure defined through field level
		- Data processing resource budgets allocated to CSC's (timing, storage, accuracy) as appropriate
		- Verified for completeness, consistency, feasibility, traceability to requirements
		Preliminary integration and test plan, draft users' manual for this applications build
*		Preliminary overall integration and test plan, acceptance test plan

This has proven to be dangerous - at a minimum I would only state "as appropriate"

→ Some CSCs very important
→ Most CSCs are not
Critical threads should provide insight into the important ones

PDW: Technical aspects of this item completed and reviewed at Preliminary Design Walkthrough

PDR1: This item completed at first applications build PDR, and updated at later PDR's